Journal of Computational and Applied Mathematics 234 (2010) 3007–3024

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

The fast quartic solver

Peter Strobach

AST-Consulting Inc., Bahnsteig 6, 94133 Röhrnbach, Germany

ARTICLE INFO

Article history:
Received 27 February 2010
Received in revised form 14 April 2010

MSC:
65E05

Keywords:
Quartic function
Polynomial factorization
Polynomial rooting
Companion matrix
Eigenvalues

ABSTRACT

A fast and highly accurate algorithm for solving quartic equations is introduced. This new algorithm is more than six times as fast and several times more accurate than the quasi-standard Companion matrix eigenvalue quartic solver. Moreover, the method is exceptionally robust in cases of extreme root spread. The new algorithm is based on a factorization of the quartic in two quadratics, which are solved in closed form. The performance key at this point is a fixed-point iteration based fitting algorithm for backward optimization of the underlying quartic-to-quadratic polynomial decomposition. Detailed experimental results confirm our claims.

1. Introduction

Consider the practical rooting of quartic polynomials or functions of the type

$$\begin{aligned} f(x) &= x^4 + ax^3 + bx^2 + cx + d \\ &= \left(x^2 + \alpha x + \beta\right)\left(x^2 + \gamma x + \delta\right) \\ &= (x - x_1)(x - x_2)(x - x_3)(x - x_4)\,. \end{aligned} \tag{1}$$

A quartic solver is an algorithm that relates a set of real or complex conjugate or mixed real/complex conjugate roots $\{x_1, x_2, x_3, x_4\}$ to a given set of real quartic coefficients $\{a, b, c, d\}$.

A closed-form algorithm or solution formula for the quartic (1) has been developed by *L.* Ferrari in the mid 16th century [1,2]. One of the most profound outlines about the quartic, its closed-form solution, applications and further readings can be found in [3], which also contains a summary or quasicode of Ferrari's algorithm suitable for implementation.

An implementation of this closed-form quartic solver shows that it has a certain roundoff error characteristic that makes it unsuitable for solving quartics with large root spread, where the root spread is defined as the ratio of the largest and the smallest root magnitude according to

$$S = \frac{|x_{\max}|}{|x_{\min}|}. \tag{2}$$

A plain closed-form quartic solver produces inferior results for the tiny roots in cases where S is large. No thorough theoretical analysis of the roundoff error characteristics of closed-form quartic solvers exists until now. These drawbacks prevented the practical application of the elegant closed-form quartic solver in the past.

E-mail address: peter_strobach@gmx.de.
URL: http://www.ast-consulting.net.

0377-0427/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.cam.2010.04.015

In practice, the problem is often bypassed and the roots of a quartic are computed as the eigenvalues of the associated coefficient companion matrix

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 0 & -d \\ 1 & 0 & 0 & -c \\ 0 & 1 & 0 & -b \\ 0 & 0 & 1 & -a \end{bmatrix}. \tag{3}$$

See [4] for details. This way the original rooting problem is posed as an unsymmetric eigenvalue problem, which is then solved using standard software for computing the eigenvalues of unsymmetric matrices.

A great amount of effort has gone into the development of algorithms and software for solving the standard unsymmetric eigenvalue problem. The most widely accepted algorithm in this area is probably the unsymmetric QR algorithm [4]. Refined software meanwhile exists for this algorithm. For instance, in terms of the Lapack subroutine `dgeev.f` [5]. This subroutine can be employed for computing the roots of $f(x)$ as the eigenvalues of $\mathbf{C}$. A special subroutine for solving quartics also exists in terms of the NAG Fortran subroutine `c02alf.f` [6]. This subroutine performs exactly like `dgeev.f` applied to the $\mathbf{C}$ of (3), but comprises some additional tools for estimating the errors of the calculated roots.

These Companion–eigenvalue type quartic solvers can handle large root spreads without problems, however at the price of excessive runtimes. For instance, `co2alf.f` requires an accumulated CPU-time of 35.4 s for solving one million quartics on a conventional dual-core Laptop computer. A plain realization of a quartic solver based on `dgeev.f` requires 26.5 s of CPU-time for accomplishing this task, while the classical closed-form quartic solver of [3] requires only 2.1 s of accumulated CPU-time for performing the same task on the same computer in the same overall environment.

Moreover, we found that the Companion–eigenvalue quartic solvers based on the unsymmetric QR algorithm produce an overall roundoff error level that is much higher than the roundoff error level that can be reached in a problem of this kind.

This overall situation has motivated the search for a new quartic solver concept, with runtimes in the range of the closed-form solver, but with generally improved accuracy in terms of low roundoff error and significantly improved robustness in cases of extreme spread.

A fixed-point iteration based fitting algorithm is introduced that adapts the two quadratic polynomial approximants of (1) with coefficients $\{\alpha, \beta, \gamma, \delta\}$ onto a given quartic with coefficients $\{a, b, c, d\}$ in the sense of an overall minimization of a fitting error. This algorithm offers some useful and interesting properties: (1) The fitting error in this problem is only a weakly nonlinear function of the coefficients $\{\alpha, \beta, \gamma, \delta\}$. Hence the basin of attraction of this iteration will be quite large. (2) The underlying Jacobian matrix consists directly of the iterated $\{\alpha, \beta, \gamma, \delta\}$ coefficients and contains no other computed elements. Hence there is no squaring and no additional error induced at this point. (3) This Jacobian matrix exhibits a special concatenated band structure that allows the development of a very fast and numerically well-conditioned updating algorithm for the $\{\alpha, \beta, \gamma, \delta\}$ coefficients. This algorithm is developed in Section 2. In Section 3, we show how the classical closed-form solver can be used as a start-up algorithm for computing the initial solution for the fast fixed-point iterations. These fixed-point iterations, on the other hand, act like a backward optimizer on the coarse estimates provided by the closed-form solver. Some special considerations are required here for a connection of these two subalgorithms. Once the coefficients of the quadratics are completely determined, the roots are obtained by conventional closed-form calculation. A large number of experimental results are discussed in Section 4. Section 5 presents the conclusions.

2. A fast fixed-point iteration for quadratic-to-quartic fitting

Consider the problem of fitting two quadratics onto one quartic. Given the quartic coefficients $\{a, b, c, d\}$, we must adjust the quadratic coefficients $\{\alpha, \beta, \gamma, \delta\}$ so that the following relation holds:

$$\begin{bmatrix} 1 \\ a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 1 & & \\ \alpha & 1 & \\ \beta & \alpha & 1 \\ & \beta & \alpha \\ & & \beta \end{bmatrix} \begin{bmatrix} 1 \\ \gamma \\ \delta \end{bmatrix} = \begin{bmatrix} 1 \\ \alpha + \gamma \\ \beta + \alpha\gamma + \delta \\ \beta\gamma + \alpha\delta \\ \beta\delta \end{bmatrix}. \tag{4}$$

In the case of a mismatch, an error $\mathbf{e}$ will occur:

$$\mathbf{e} = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \end{bmatrix} = \begin{bmatrix} a - \alpha - \gamma \\ b - \beta - \alpha\gamma - \delta \\ c - \beta\gamma - \alpha\delta \\ d - \beta\delta \end{bmatrix}. \tag{5}$$

The Jacobian matrix of this error can be established as follows:

$$\mathbf{F} = \begin{bmatrix} \dfrac{\partial}{\partial\alpha}\mathbf{e} & \dfrac{\partial}{\partial\beta}\mathbf{e} & \dfrac{\partial}{\partial\gamma}\mathbf{e} & \dfrac{\partial}{\partial\delta}\mathbf{e} \end{bmatrix} = -\begin{bmatrix} 1 & 0 & 1 & 0 \\ \gamma & 1 & \alpha & 1 \\ \delta & \gamma & \beta & \alpha \\ 0 & \delta & 0 & \beta \end{bmatrix}. \tag{6}$$

This matrix has an interesting concatenated band structure. We will exploit this fact later. Introduce the parameter vector $\mathbf{p}$ of the quadratic polynomial coefficients as follows:

$$\mathbf{p} = \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix}. \tag{7}$$

A linear Taylor series expansion of the coefficient error can be established:

$$\bar{\mathbf{e}}(\mathbf{p}) = \mathbf{e}(\mathbf{p}_0) + \mathbf{F}(\mathbf{p}_0)\,(\mathbf{p} - \mathbf{p}_0)\,, \tag{8}$$

where $\mathbf{p}_0$ denotes any initial guess of the quadratic polynomial coefficients. A fixed-point iteration can be deduced from (8) via setting $\bar{\mathbf{e}}(\mathbf{p}) = \mathbf{0}$. This yields:

$$\mathbf{p} = \mathbf{p}_0 - \mathbf{F}^{-1}(\mathbf{p}_0)\mathbf{e}(\mathbf{p}_0), \tag{9}$$

or equivalently

$$\mathbf{p} = \mathbf{p}_0 + \mathbf{y}, \tag{10}$$

where

$$-\mathbf{F}\mathbf{y} = \mathbf{e}. \tag{11}$$

This is a system of linear equations for the updating vector $\mathbf{y}$ using the concatenated band matrix $\mathbf{F}$ of (6). Suppose that we wish to solve this system of linear equations via LU-factorization of $-\mathbf{F}$ followed by standard forward/backsubstitution. We write:

$$-\mathbf{F} = \mathbf{L}\mathbf{U}, \tag{12}$$

and solve via forward/backsubstitution as follows:

$$\mathbf{L}\mathbf{x} = \mathbf{e} \rightarrow \mathbf{x}, \tag{13a}$$

$$\mathbf{U}\mathbf{y} = \mathbf{x} \rightarrow \mathbf{y}. \tag{13b}$$

The LU-factorization of (12) can be evaluated conveniently in closed form. Hereby, the special structure of $\mathbf{F}$, as displayed in (6), is exploited. We write:

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ \gamma & 1 & \alpha & 1 \\ \delta & \gamma & \beta & \alpha \\ 0 & \delta & 0 & \beta \end{bmatrix} = \begin{bmatrix} L_{11} & & & \\ L_{21} & L_{22} & & \\ L_{31} & L_{32} & L_{33} & \\ L_{41} & L_{42} & L_{43} & L_{44} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & U_{13} & U_{14} \\ & U_{22} & U_{23} & U_{24} \\ & & U_{33} & U_{34} \\ & & & U_{44} \end{bmatrix}. \tag{14}$$

Assume $L_{11} = 1$. Then $L_{11}U_{11} = 1$ and consequently, $U_{11} = 1$. Continue with $L_{11}U_{12} = 0$. This yields $U_{12} = 0$. Proceed in the same fashion with $L_{11}U_{13} = 1$ which gives $U_{13} = 1$ and $L_{11}U_{14} = 0$ yielding $U_{14} = 0$.

Continue with row #2. $L_{21}U_{11} = \gamma$ yields $L_{21} = \gamma$. $L_{21}U_{12} + L_{22}U_{22} = 1$ yields $U_{22} = 1$, where we assumed $L_{22} = 1$. Consequently, $L_{21}U_{13} + L_{22}U_{23} = \alpha$ yields

$$U_{23} = \alpha - \gamma. \tag{15}$$

Moreover, $L_{21}U_{14} + L_{22}U_{24} = 1$ yields $U_{24} = 1$.

Proceed with row #3. $L_{31}U_{11} = \delta$ gives $L_{31} = \delta$. Likewise, $L_{31}U_{12} + L_{32}U_{22} = \gamma$ gives $L_{32} = \gamma$. An evaluation of $L_{31}U_{13} + L_{32}U_{23} + L_{33}U_{33} = \beta$ yields

$$L_{33}U_{33} = \beta - \delta + \gamma(\gamma - \alpha). \tag{16}$$

Finally $L_{31}U_{14} + L_{32}U_{24} + L_{33}U_{34} = \alpha$ results in

$$L_{33}U_{34} = \alpha - \gamma. \tag{17}$$

Terminate with an evaluation of row #4. $L_{41}U_{11} = 0$ yields $L_{41} = 0$ and $L_{41}U_{12} + L_{42}U_{22} = \delta$ yields $L_{42} = \delta$. Consequently $L_{41}U_{13} + L_{42}U_{23} + L_{43}U_{33} = 0$ yields:

$$L_{43}U_{33} = \delta(\gamma - \alpha). \tag{18}$$

Finally $L_{41}U_{14} + L_{42}U_{24} + L_{43}U_{34} + L_{44}U_{44} = \beta$ yields

$$L_{43}U_{34} + L_{44}U_{44} = \beta - \delta. \tag{19}$$

There remains a system of 4 coupled equations in terms of (16), (17), (18) and (19) for the 6 unknown quantities L_{33}, U_{33}, L_{44}, U_{44}, L_{43} and U_{34}. Apparently, two of these parameters can be chosen arbitrarily. The others are then determined by the

equations. In LU-decompositions, one usually fixes the main diagonal of the L-matrix to all ones. Therefore, we set $L_{33} = 1$ and $L_{44} = 1$. Then (17) immediately yields:

$$U_{34} = \alpha - \gamma = U_{23}. \tag{20}$$

Consequently,

$$U_{33} = \beta - \delta - \gamma U_{23}. \tag{21}$$

Now we can solve (18) for L_{43}. This yields:

$$L_{43} = -\frac{\delta U_{23}}{U_{33}}. \tag{22}$$

Finally, (19) determines U_{44} as follows:

$$U_{44} = \beta - \delta - L_{43}U_{23}, \tag{23}$$

and the desired LU-factorization is complete. The decomposition (14) can be rewritten explicitly as follows:

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ \gamma & 1 & \alpha & 1 \\ \delta & \gamma & \beta & \alpha \\ 0 & \delta & 0 & \beta \end{bmatrix} = \begin{bmatrix} 1 & & & \\ \gamma & 1 & & \\ \delta & \gamma & 1 & \\ 0 & \delta & L_{43} & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ & 1 & U_{23} & 1 \\ & & U_{33} & U_{23} \\ & & & U_{44} \end{bmatrix}, \tag{24}$$

with U_{23} as defined in (15), U_{33} as defined in (21), L_{43} as defined in (22) and U_{44} as defined in (23).

Next consider the forward substitution step (13a). An explicit evaluation of this system yields the following expressions for the components $\{x_1, x_2, x_3, x_3\}$ of the auxiliary vector $\mathbf{x}$ in (13a). (Notice that these components have nothing to do with the roots of (1)):

$$x_1 = e_1, \tag{25a}$$

$$x_2 = e_2 - \gamma x_1, \tag{25b}$$

$$x_3 = e_3 - \delta x_1 - \gamma x_2, \tag{25c}$$

$$x_4 = e_4 - \delta x_2 - L_{43}x_3. \tag{25d}$$

These components are now used in the backsubstitution step (13b) to determine the components $\{y_1, y_2, y_3, y_3\}$ of the desired updating vector $\mathbf{y}$ as follows:

$$y_4 = \frac{x_4}{U_{44}}, \tag{26a}$$

$$y_3 = \frac{x_3 - U_{23}y_4}{U_{33}}, \tag{26b}$$

$$y_2 = x_2 - U_{23}y_3 - y_4, \tag{26c}$$

$$y_1 = x_1 - y_3. \tag{26d}$$

Table 1 is a summary of this fast algorithm for iterative refinement of the coefficient set $\{\alpha, \beta, \gamma, \delta\}$ from a given initial guess $\{\alpha_0, \beta_0, \gamma_0, \delta_0\}$. We can easily count that this algorithm requires only 14 multiplications and 3 divisions per update or refinement step of the $\{\alpha, \beta, \gamma, \delta\}$ coefficient set. Usually only a few of these iterations will be required, because this algorithm converges extremely rapidly as a consequence of the fact that $\mathbf{e}$ is only a weakly nonlinear function in the $\{\alpha, \beta, \gamma, \delta\}$ coefficients (recall (5)). The iteration is usually terminated before reaching the maximum iterations counter m via monitoring of an overall residual function ϵ defined as follows:

$$\epsilon = |e_1| + |e_2| + |e_3| + |e_4|. \tag{27}$$

This fitting residual ϵ will converge to very small values or converges to zero (perfect fit). In cases of a nonzero ϵ, we can observe that this quantity shows a limit cycle behavior. These limit cycles are a reliable convergence criterion. In cases where ϵ is not perfectly nulled, we apply some simple forms of "pattern recognition" for the detection of these limit cycles as a test of convergence.

2.1. Multiple roots

Until now, we assumed the general case of four simple roots. However, there can be cases with multiple roots. Some of these cases can be detected very easily and the multiple roots can be calculated conveniently in closed form. This is regarded a kind of exception handling. The following cases can occur:

(1) One double root and two simple roots.
(2) Two double roots.
(3) One quartic root (4-fold root).
(4) One cubic root and one simple root.

Table 1
Fast fixed-point type iterative refinement algorithm or backward optimizer for the coefficient set $\{\alpha, \beta, \gamma, \delta\}$. Equations numbered as they appear in the text.

Initialize: $\begin{cases} \alpha := \alpha_0 \\ \beta := \beta_0 \\ \gamma := \gamma_0 \\ \delta := \delta_0 \end{cases}$

$$\begin{bmatrix} e_1 = a - \alpha - \gamma & (5) \\ e_2 = b - \beta - \alpha\gamma - \delta & (5) \\ e_3 = c - \beta\gamma - \alpha\delta & (5) \\ e_4 = d - \beta\delta & (5) \end{bmatrix}$$

FOR $\mu = 1, 2, 3, \ldots, m$ iterate:

$$\begin{bmatrix} U_{23} = \alpha - \gamma & (15) \\ U_{33} = \beta - \delta - \gamma U_{23} & (21) \\ L_{43} = -\dfrac{\delta U_{23}}{U_{33}} & (22) \\ U_{44} = \beta - \delta - L_{43}U_{23} & (23) \\ x_1 = e_1 & (25a) \\ x_2 = e_2 - \gamma x_1 & (25b) \\ x_3 = e_3 - \delta x_1 - \gamma x_2 & (25c) \\ x_4 = e_4 - \delta x_2 - L_{43}x_3 & (25d) \\ y_4 = \dfrac{x_4}{U_{44}} & (26a) \\ y_3 = \dfrac{x_3 - U_{23}y_4}{U_{33}} & (26b) \\ y_2 = x_2 - U_{23}y_3 - y_4 & (26c) \\ y_1 = x_1 - y_3 & (26d) \\ \alpha \leftarrow \alpha + y_1 & (10) \\ \beta \leftarrow \beta + y_2 & (10) \\ \gamma \leftarrow \gamma + y_3 & (10) \\ \delta \leftarrow \delta + y_4 & (10) \\ e_1 = a - \alpha - \gamma & (5) \\ e_2 = b - \beta - \alpha\gamma - \delta & (5) \\ e_3 = c - \beta\gamma - \alpha\delta & (5) \\ e_4 = d - \beta\delta & (5) \end{bmatrix}$$

Case (1) requires no exception handling. Cases (2) and (3) fall into the same category and can be detected and treated by the same exception handler. A second exception handler is required for case (4). These exception handlers are next introduced.

Cases (2) and (3) are characterized by two perfectly identical quadratics. This results in the following parameter identities:

$$\gamma = \alpha, \tag{28a}$$

$$\delta = \beta. \tag{28b}$$

Clearly, this would cause a rank-drop of the Jacobian matrix (6) and the iterative algorithm of Table 1 would collapse. Fortunately, the case is easily detected beforehand by looking at the quartic coefficients with some care. Recall (4). In the case (28a) and (28b), we obtain the special relations

$$a = 2\alpha, \tag{29a}$$

$$b = \alpha^2 + 2\beta, \tag{29b}$$

$$c = 2\alpha\beta, \tag{29c}$$

$$d = \beta^2. \tag{29d}$$

One option here is to determine α and β from the first two equations in this set. This yields:

$$\alpha = \frac{a}{2}, \tag{30a}$$

$$\beta = \frac{b - \alpha^2}{2}. \tag{30b}$$

These values for α and β must perfectly satisfy the remaining equations (29c) and (29d). To check this, two error variables are introduced:

$$\epsilon_1 = c - 2\alpha\beta, \tag{31a}$$

$$\epsilon_2 = d - \beta^2. \tag{31b}$$

A test is performed, if these two error variables ϵ_1 and ϵ_2 are both *perfectly nulled*. If this condition is fulfilled, we simply obtain the two double roots as the roots of the quadratic equation $q(x) = x^2 + \alpha x + \beta$. It can happen that $q(x)$ itself has a double root. This double root is then a 4-fold root of the given quartic.

Another exception handles the case of a cubic root. This case can be detected in a similar fashion as seen before. Suppose that x_1 is a cubic root and x_2 is a simple root of the quartic $f(x)$ in formula (1). Then it is readily verified that the quartic coefficients can be expressed in terms of these roots as follows:

$$a = -3x_1 - x_2, \tag{32a}$$

$$b = 3x_1\,(x_1 + x_2)\,, \tag{32b}$$

$$c = -x_1^2\,(x_1 + 3x_2)\,, \tag{32c}$$

$$d = x_1^3 x_2. \tag{32d}$$

Again, we have here four equations for only two unknowns. The first two equations in this set can be used to establish the following quadratic equation for the unknown cubic root x_1 as follows.

$$x_1^2 + \frac{a}{2}x_1 + \frac{b}{6} = 0. \tag{33}$$

For *each* of the generally two solutions of this quadratic equation, we compute a corresponding solution x_2,

$$x_2 = -a - 3x_1, \tag{34}$$

and obtain two solution pairs $\{x_1, x_2\}_1$ and $\{x_1, x_2\}_2$. Only one of these two solution pairs satisfies (32c) and (32d) and is hence the desired true solution. The following error variables ϵ_1 and ϵ_2 are *perfectly nulled* by the true solution:

$$\epsilon_1 = c + x_1^2\,(x_1 + 3x_2)\,, \tag{35a}$$

$$\epsilon_2 = d - x_1^3 x_2. \tag{35b}$$

3. A start-up from closed-form solution concepts

In the general case, the calculations of Table 1 are activated. The algorithm of Table 1 constitutes a refinement algorithm that requires an initial solution for the coefficients $\{\alpha_0, \beta_0, \gamma_0, \delta_0\}$ of the two quadratics as input. Basically, any desired method for obtaining this initial solution can be used. However, iterative methods are unsuitable because of unclear convergence characteristics and runtimes.

A worthwhile option are closed-form solution concepts. One first idea in this context would be the closed-form calculation of the desired coefficient set $\{\alpha_0, \beta_0, \gamma_0, \delta_0\}$ directly from the given quartic coefficients in the classical way by the method of radicals. This is possible and the solution is described in [3]. A particularly appealing presentation of the main results can also be found in Dr. Math@Drexel [7].

The method begins with the classical depression of the given quartic. Then there exists a decomposition of the depressed quartic into two quadratics of only two variables. These two variables can be determined by solving a cubic in closed form. Then there exists a suitable transformation that allows these two variables of the quadratic decomposition of the depressed quartic to be transformed back into the desired coefficients $\{\alpha_0, \beta_0, \gamma_0, \delta_0\}$.

This constitutes altogether the most straight way to obtain an initial solution. Unfortunately, our tests revealed that the underlying calculations are extremely sensitive to roundoff error. In cases of extreme spread, the so-obtained initial values appear totally deteriorated. This solution is therefore of no value for a practical application in this context.

A second option exists in terms of the closed-form summary of Ferrari's algorithm as listed in [3]. This algorithm computes directly the roots and hence does more than required. But we shall soon realize that this first-glance redundancy will be quite of some advantage in our application.

We implemented this version of Ferrari's algorithm, as listed in [3], with the leading (top) coefficient scaled to value of 1. In this algorithm, all roots are generally declared as complex variables. Real roots appear as special cases with vanishing imaginary component. The algorithm produces the roots in disorder. They are ordered afterwards in descending order of magnitude. Table 2 shows the cases of real and complex conjugate root combinations that can occur in such a set of ordered roots. Each root is displayed in its Euler form as $x_k = r_k e^{j\varphi_k}$, where $r_k = |x_k|$ and "j" denotes the imaginary unit.

There are basically three cases. The case of four real roots, displayed as case 1 in Table 2, the case of two complex conjugate root pairs, displayed as case 5 in Table 2, and finally the case of two real roots and one complex conjugate root pair. This latter case appears in terms of three sub-cases depending on the magnitude of the complex conjugate root pair relative to the magnitudes of the two real roots. The complex conjugate root pair may appear at the bottom (case 2), in the middle (case 3), or may appear on top of such an ordered root set (case 4).

Unfortunately, though, we will hardly be able to identify any of these cases in the practically computed roots of a closed-form solver, because in the practice of computing roots at larger root spreads, all the computed roots will show some more

Table 2
Possible root combinations for a quartic equation with real coefficients and ordered roots $|x_1| \geq |x_2| \geq |x_3| \geq |x_4|$.

	Case 1	Case 2	Case 3	Case 4	Case 5
x_1	r_1	r_1	r_1	$r_1 e^{j\varphi_1}$	$r_1 e^{j\varphi_1}$
x_2	r_2	r_2	$r_2 e^{j\varphi_2}$	$r_1 e^{-j\varphi_1}$	$r_1 e^{-j\varphi_1}$
x_3	r_3	$r_3 e^{j\varphi_3}$	$r_2 e^{-j\varphi_2}$	r_3	$r_2 e^{j\varphi_2}$
x_4	r_4	$r_3 e^{-j\varphi_3}$	r_4	r_4	$r_2 e^{-j\varphi_2}$

or less pronounced imaginary components caused by roundoff error, even in cases where all roots are ideally real. Hence there will be no way to identify safely any of these cases of Table 2 in practically computed root sets.

One more practical observation can be made. The closed-form solver algorithm of [3] produces the dominant root x_1 with a very high relative accuracy. This dominant root often appears even perfectly ideal (without any roundoff error). The subdominant roots appear increasingly corrupted by roundoff error. In critical cases, the minor roots x_3 and x_4 may appear totally deteriorated. We will demonstrate this in the following experimental section.

From these practical observations, we can conclude that only the two dominant (leading) roots x_1 and x_2 of the closed-form solver will be of some value for our purposes. We can use these roots for computing the initial values α_0 and β_0 according to:

$$\begin{aligned}(x - x_1)(x - x_2) &= x^2 - (x_1 + x_2)x + x_1 x_2 \\ &= x^2 + \alpha_0 x + \beta_0,\end{aligned} \tag{36}$$

yielding

$$\alpha_{01} = -\mathrm{re}\{x_1 + x_2\}, \tag{37a}$$
$$\beta_{01} = \mathrm{re}\{x_1 x_2\}, \tag{37b}$$

where $\mathrm{re}\{\cdot\}$ extracts the real part of the argument. Clearly, this operator is ideally superfluous, however, is indispensable here because of possible spurious imaginary roundoff error components in the root estimates x_1 and x_2 produced by the closed-form solver.

At this point, we realize that this strategy of computing the initial estimates α_0 and β_0 is not applicable in case 3 of Table 2, because in that case, the dominant roots x_1 and x_2 will not recombine to valid real coefficients α_0 and β_0 in the ideal case. But this is a necessary side-condition in our concept. Hence in case 3, there is no other way but computing the coefficient estimates from the roots x_2 and x_3 according to:

$$\alpha_{02} = -\mathrm{re}\{x_2 + x_3\}, \tag{38a}$$
$$\beta_{02} = \mathrm{re}\{x_2 x_3\}. \tag{38b}$$

We realize the difficulty at this point, that we cannot generally identify from the computed roots to which one of the 5 cases in Table 2 they belong. As a consequence of roundoff error, there will be no save way to distinguish whether (37a,b) or (38a,b) will be the correct case yielding valid real coefficient estimates.

Therefore, the only way to handle this case is to work with *two* initial coefficient estimates, namely, the set $\{\alpha_{01}, \beta_{01}\}$ according to (37a) and (37b), and the set $\{\alpha_{02}, \beta_{02}\}$ according to (38a) and (38b) in *parallel*. One of these two sets will in any event reflect a valid case. We bypass the decision about which one of these two initializations will be the correct one at this point, because we could not safely justify such a decision here.

The save way at this point lies in the decision of running the backward optimizer of Table 1 two times *individually* in parallel, with two different initial settings, namely $\{\alpha_{01}, \beta_{01}\}$ and $\{\alpha_{02}, \beta_{02}\}$. Either the one, or the other, or both will converge to the desired refined $\{\alpha, \beta\}$ coefficients. This is called a *tandem iteration*, because two realizations of the algorithm of Table 1 are operated individually in parallel, with different starting points on the coefficients. We leave it entirely up to this tandem iteration to find the right way. This method safely works and we avoided a heuristical untrustworthy decision about the root case that we obtain from the closed-form start-up root finder.

This is also a method that is affordable in terms of the overall amount of computations, because we have seen that the algorithm of Table 1 is very fast and requires only a few iterations to converge from a good initial starting point. In our concept, we offer the algorithm two starting points. At least one of them will result in a rapid convergence and will terminate the tandem iteration in a single or in a few iterations, as we shall demonstrate in the experimental section.

Until now, we only discussed the generation of the initial values $\{\alpha_{01}, \beta_{01}\}$ and $\{\alpha_{02}, \beta_{02}\}$ from the coarse root estimates of the closed-form quartic solver. The algorithm of Table 2, however, additionally requires the corresponding complementary coefficients $\{\gamma_{01}, \delta_{01}\}$ and $\{\gamma_{02}, \delta_{02}\}$ for a complete initialization. How are these complementary coefficients computed? They cannot be obtained directly from the closed-form quartic solver roots, because we exploited all the trustworthy information in these roots already for computing the $\{\alpha, \beta\}$ initial estimates. Hence we must resort to the given quartic coefficient set itself for computing this missing complementary information. For this purpose, recall (5). The missing $\{\gamma_0, \delta_0\}$ information

is apparently given by the following overdetermined system of linear equations:

$$\mathbf{e}_{LS} = \begin{bmatrix} a-\alpha_0 \\ b-\beta_0 \\ c \\ d \end{bmatrix} - \begin{bmatrix} 1 & \\ \alpha_0 & 1 \\ \beta_0 & \alpha_0 \\ & \beta_0 \end{bmatrix} \begin{bmatrix} \gamma_0 \\ \delta_0 \end{bmatrix}, \tag{39}$$

where $\mathbf{e}_{LS}$ denotes a least squares error vector. This system can be evaluated for the two cases of interest, namely $\alpha_0 = \alpha_{01},\ \beta_0 = \beta_{01}$ and $\alpha_0 = \alpha_{02},\ \beta_0 = \beta_{02}$. The solution of (39) then yields the desired $\gamma_0 = \gamma_{01},\ \delta_0 = \delta_{01}$ and $\gamma_0 = \gamma_{02},\ \delta_0 = \delta_{02}$ initial coefficients, respectively.

This little least squares problem can be solved in many ways. We describe here our solution, which we used in the experimental implementation of the algorithm. It is considerably clear that we can set up a system of normal equations for this case as follows:

$$\begin{bmatrix} \Phi_1 & \Phi_2 \\ \Phi_2 & \Phi_1 \end{bmatrix} \begin{bmatrix} \gamma_0 \\ \delta_0 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}, \tag{40}$$

where

$$\Phi_1 = 1 + \alpha_0^2 + \beta_0^2, \tag{41a}$$

$$\Phi_2 = \alpha_0(1+\beta_0), \tag{41b}$$

and

$$c_1 = a - \alpha_0 + \alpha_0(b-\beta_0) + \beta_0 c, \tag{42a}$$

$$c_2 = b - \beta_0 + \alpha_0 c + \beta_0 d. \tag{42b}$$

Introduce a closed-form Cholesky decomposition as follows:

$$\begin{bmatrix} \Phi_1 & \Phi_2 \\ \Phi_2 & \Phi_1 \end{bmatrix} = \begin{bmatrix} L_1 & 0 \\ L_3 & L_2 \end{bmatrix} \begin{bmatrix} L_1 & L_3 \\ 0 & L_2 \end{bmatrix} = \begin{bmatrix} (L_1^2) & (L_1 L_3) \\ (L_1 L_3) & (L_2^2 + L_3^2) \end{bmatrix}. \tag{43}$$

A comparison of elements in (43) yields:

$$L_1 = \sqrt{\Phi_1}, \tag{44a}$$

$$L_3 = \frac{\Phi_2}{L_1}, \tag{44b}$$

$$L_2 = \sqrt{\Phi_1 - \frac{\Phi_2}{\Phi_1}\Phi_2}. \tag{44c}$$

Finally, consider the necessary forward/backsubstitution step in this algorithm:

$$\begin{bmatrix} L_1 & 0 \\ L_3 & L_2 \end{bmatrix} \begin{bmatrix} L_1 & L_3 \\ 0 & L_2 \end{bmatrix} \begin{bmatrix} \gamma_0 \\ \delta_0 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}. \tag{45}$$

First solve

$$\begin{bmatrix} L_1 & 0 \\ L_3 & L_2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \tag{46}$$

for the intermediate variables y_1 and y_2. This yields:

$$y_1 = \frac{c_1}{L_1}, \tag{47a}$$

$$y_2 = \frac{c_2 - y_1 L_3}{L_2}. \tag{47b}$$

Then solve

$$\begin{bmatrix} L_1 & L_3 \\ 0 & L_2 \end{bmatrix} \begin{bmatrix} \gamma_0 \\ \delta_0 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \tag{48}$$

for the desired coefficients γ_0 and δ_0:

$$\delta_0 = \frac{y_2}{L_2}, \tag{49a}$$

$$\gamma_0 = \frac{y_1 - \delta_0 L_3}{L_1}. \tag{49b}$$

Table 3
Fast least squares algorithm for computing the coefficients $\{\gamma_0, \delta_0\}$ for a given set $\{\alpha_0, \beta_0\}$. Equations numbered as they appear in the text.

$$\Phi_1 = 1 + \alpha_0^2 + \beta_0^2 \quad (41a)$$
$$\Phi_2 = \alpha_0(1 + \beta_0) \quad (41b)$$
$$c_1 = a - \alpha_0 + \alpha_0(b - \beta_0) + \beta_0 c \quad (42a)$$
$$c_2 = b - \beta_0 + \alpha_0 c + \beta_0 d \quad (42b)$$
$$L_1 = \sqrt{\Phi_1} \quad (44a)$$
$$L_3 = \frac{\Phi_2}{L_1} \quad (44b)$$
$$L_2 = \sqrt{\Phi_1 - \frac{\Phi_2}{\Phi_1}\Phi_2} \quad (44c)$$
$$y_1 = \frac{c_1}{L_1} \quad (47a)$$
$$y_2 = \frac{c_2 - y_1 L_3}{L_2} \quad (47b)$$
$$\delta_0 = \frac{y_2}{L_2} \quad (49a)$$
$$\gamma_0 = \frac{y_1 - \delta_0 L_3}{L_1} \quad (49b)$$

Table 3 is a summary of this algorithm for computing the complementary γ_0 and δ_0 initial information from the given α_0 and β_0. Of course, this algorithm is also operated in tandem configuration for the two different initial estimates of α and β.

The complete fast quartic solver algorithm of this paper can be summarized as follows:

(1) Given a set of quartic coefficients $\{a, b, c, d\}$, compute the initial root estimates $\{x_1, x_2, x_3, x_4\}$ using the classical closed-form quartic solver of [3]. Order these root estimates in descending order of magnitude.
(2) Compute the initial coefficient estimates $\{\alpha_{01}, \beta_{01}\}$ and $\{\alpha_{02}, \beta_{02}\}$ according to (37a), (37b) and (38a), (38b).
(3) Apply the fast least squares algorithm of Table 3 to each of these coefficient sets in order to obtain the complementary coefficients $\{\gamma_{01}, \delta_{01}\}$ and $\{\gamma_{02}, \delta_{02}\}$.
(4) Run two individual realizations of the backward optimizer of Table 1 with initial coefficient sets $\{\alpha_{01}, \beta_{01}, \gamma_{01}, \delta_{01}\}$ and $\{\alpha_{02}, \beta_{02}, \gamma_{02}, \delta_{02}\}$ (tandem iteration). After each iteration, calculate criterion ϵ according to (27) individually for each of the two iterated coefficient sets. Compare these ϵ's with the ϵ's of the 4 previous iterations which are saved on individual stacks. In each of the two individual algorithms, decide whether the actual ϵ is either perfectly zero or is exactly equal to one of the 4 stacked previous ϵ's. If this condition is fulfilled, perfect convergence or a limit cycle in one of the two individual backward optimizers is detected and both backward optimizers are stopped. The coefficient set of the perfectly convergent or limit cycling backward optimizer is selected for final rooting. If this test fails, both backward optimizers run up to a maximum iterations count m. If the optimization process exits after m iterations, the coefficient set with the smallest ϵ is selected for final rooting.
(5) Calculate the roots of the two quadratics given by the refined coefficients $\{\alpha, \beta\}$ and $\{\gamma, \delta\}$ using the closed-form solution formula of [8, Chapter 5.6].

4. Computer experiments

The purpose of this section is to visualize the performance of the new fast quartic solver in comparison with the plain closed-form solver of [3], and the Companion/eigenvalue quartic solver using either `dgeev.f` or `c02alf.f`, which both produce perfectly identical results. Results of two types of experiments are shown: (1) Long-term statistical tests, where the algorithms are operated on randomly generated root sets. One million random root sets are examined in each of these tests. (2) Individual experiments using selected root configurations with extreme spread and/or surprising results.

All implementations are in Fortran 77 style compiled using the Intel vectorizing Fortran compiler version 11 on a computer Acer TravelMate 6592 G with Intel Core 2 Duo processor T9300 (2.5 GHz, 800 MHz FSB, 6 MB L2 cache). The algorithms are implemented in double precision, the surrounding evaluation software is implemented in quadruple precision.

4.1. Long-term statistical tests

Results of long-term statistical tests are shown here where the algorithms are operated on randomly generated root sets. One million root sets are examined in each of these experiments. Three cases are distinguished: The case of all real roots, the case of two real roots and one complex conjugate root pair, which may appear in any of the three locations 2, 3, or 4 in the magnitude ranking shown in Table 2. Finally, the case of two complex conjugate root pairs.

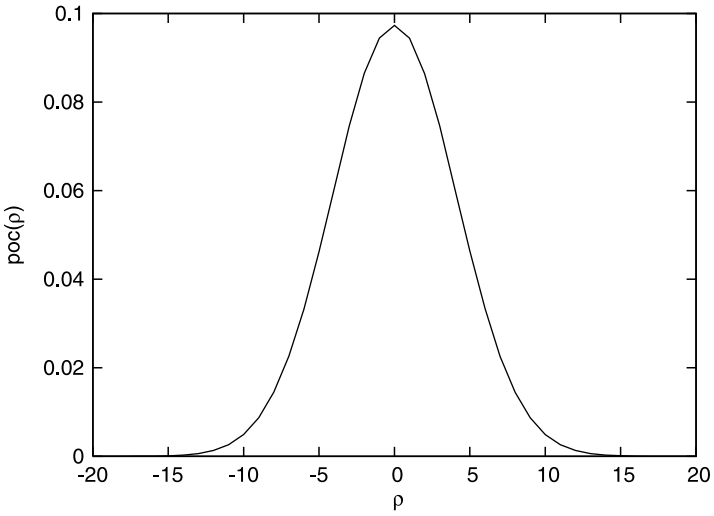

Fig. 1. Probability of occurrence of a random variable ρ.

In this experiment, we monitor the mean square root error defined as follows:

$$E = \sqrt{\sum_{k=1}^{4} |x_k - \hat{x}_k|^2}, \tag{50}$$

where the x_k's are the true roots and the $\hat{x}_k$'s are the computed roots using one of the algorithms under comparison. The true roots are modeled as statistically independent samples of a zero-mean approximately Gaussian distributed random process $\{\rho\}$ according to the distribution shown in Fig. 1.

In the case of all real roots, these roots are constructed from the random samples $\{\rho\}$ as follows:

$$\begin{aligned} x_1 &= \text{cmplx}(\rho_1, 0.0) \\ x_2 &= \text{cmplx}(\rho_2, 0.0) \\ x_3 &= \text{cmplx}(\rho_3, 0.0) \\ x_4 &= \text{cmplx}(\rho_4, 0.0). \end{aligned} \tag{51}$$

In the case of two real roots and one complex conjugate root pair, we construct:

$$\begin{aligned} x_1 &= \text{cmplx}(\rho_1, 0.0) \\ x_2 &= \text{cmplx}(\rho_2, 0.0) \\ x_3 &= \text{cmplx}(\rho_3, \rho_4) \\ x_4 &= \text{cmplx}(\rho_3, -\rho_4). \end{aligned} \tag{52}$$

Notice that the complex conjugate root pair $x_{3,4}$ can have any magnitude here as a consequence of the statistically independent nature of the random variables $\{\rho\}$. Finally, we construct the root sets with two complex conjugate root pairs as follows:

$$\begin{aligned} x_1 &= \text{cmplx}(\rho_1, \rho_2) \\ x_2 &= \text{cmplx}(\rho_1, -\rho_2) \\ x_3 &= \text{cmplx}(\rho_3, \rho_4) \\ x_4 &= \text{cmplx}(\rho_3, -\rho_4). \end{aligned} \tag{53}$$

These root sets are represented in quadruple precision in the test programs. The quartic coefficients are then calculated (in quadruple precision) according to:

$$a = -(x_1 + x_2 + x_3 + x_4), \tag{54a}$$

$$b = x_1x_2 + (x_1 + x_2)(x_3 + x_4) + x_3x_4, \tag{54b}$$

$$c = -x_1x_2(x_3 + x_4) - x_3x_4(x_1 + x_2), \tag{54c}$$

$$d = x_1x_2x_3x_4. \tag{54d}$$

One million individual realizations of this quartic coefficient set $\{a, b, c, d\}$ are used in each of the following experiments.

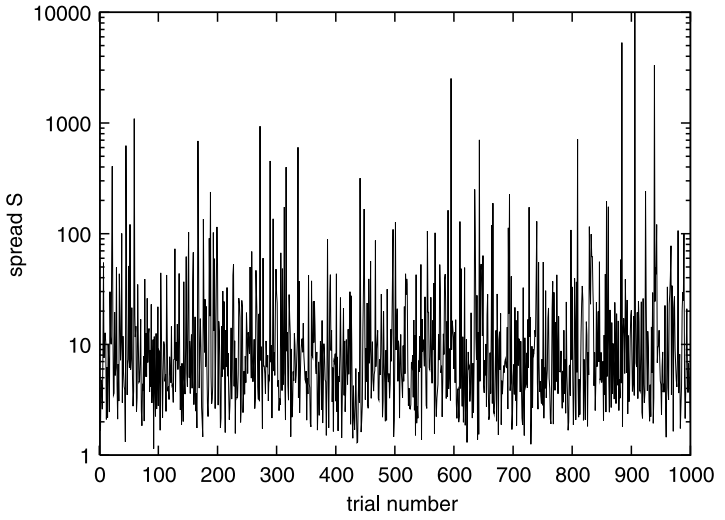

Fig. 2. Root spread S in 1000 trial runs with 4 real random roots drawn from the poc(ρ) as shown in Fig. 1.

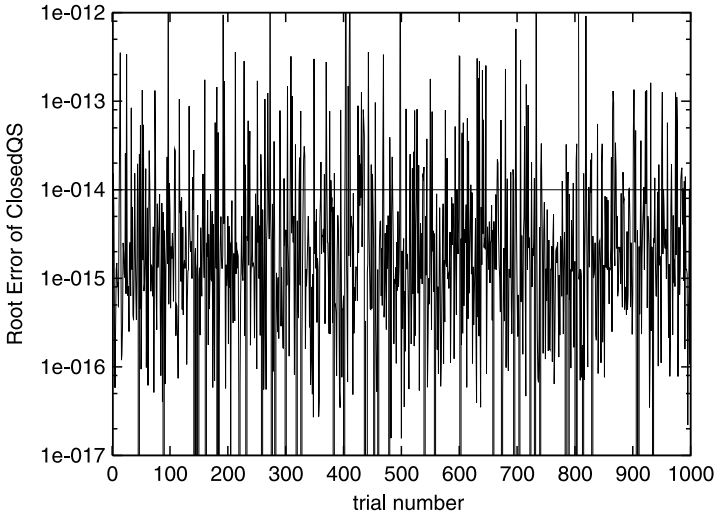

Fig. 3. Mean square root error of the closed-form quartic solver (ClosedQS) in 1000 trial runs using statistically independent random roots (4 real roots). Horizontal line at 1e−14 displayed for orientation purposes. Approximately 5% of all errors are perfectly zero.

Fig. 2 shows the spread S as defined in (2) for the first 1000 trials in the first experiment comprising one million trials with all real roots (51). We can see that as a consequence of the random nature of the roots, the spread values can be quite large.

Figs. 3–5 show the mean square root errors according to (50), as obtained in the first 1000 trials of the all real roots experiment for the three algorithms: The closed-form quartic solver (ClosedQS) of [3], the Companion matrix eigenvalue solver based on Lapack subroutine `dgeev.f` (CompQS), and the fast quartic solver as developed in this paper (FastQS). Notice again, that FastQS uses the results of ClosedQS as initial values for the backward refinement step.

We observe from Fig. 3, that the mean square root errors of ClosedQS appear lower in average than the mean square root errors of CompQS as displayed in Fig. 4. Some mean square root errors in Fig. 3 are apparently even lower than the lowest displayed value of $E = 1e{-}17$. These errors are all perfectly zero. This closed-form solver produces a perfect (error-free) reconstruction of the roots in approximately 5% of all cases over one million trials. The CompQS is lacking this partly perfect reconstruction capability. We can see this from Fig. 4. Apparently, this technique based on unsymmetric eigenvalue calculation is hampered by a lower bound of error level located approximately at $E = 1e{-}15$. This is actually much higher than the amount of accuracy that can be reached in a problem of this kind.

This becomes apparent from an inspection of Fig. 5, which shows the mean square root error for the FastQS algorithm as proposed in this paper. In this case, over 50% of all errors drop below $E = 1e{-}17$. These errors are all perfectly zero, as our detailed evaluation revealed. There is quite an impressive difference in the attainable accuracy between CompQS (Fig. 4) and FastQS (Fig. 5).

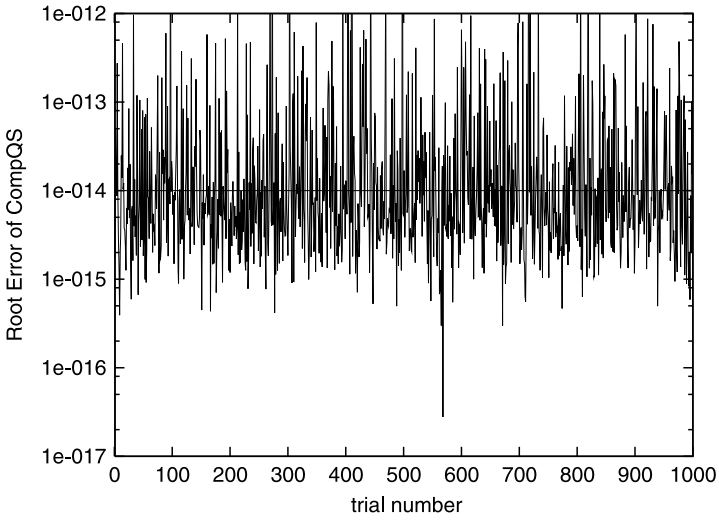

Fig. 4. Mean square root error of the Companion/eigenvalue quartic solver (CompQS) in 1000 trial runs using statistically independent random roots (4 real roots). Horizontal line at 1e−14 displayed for orientation purposes.

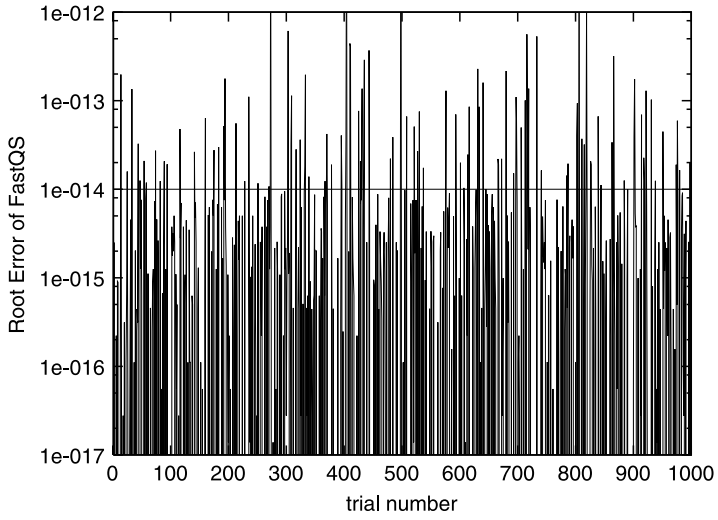

Fig. 5. Mean square root error of the fast quartic solver (FastQS) in 1000 trial runs using statistically independent random roots (4 real roots). Horizontal line at 1e−14 displayed for orientation purposes. Over 50% of all errors are perfectly zero.

This becomes even more apparent by looking at the statistical evaluation of these error tracks for the individual algorithms as shown in Fig. 6. This plot displays the estimated probability (based on the evaluation of one million independent trials) that a mean square root error exceeds a certain error level. Consider the lowest displayed error level at $E = 1e{-}17$. We can see from Fig. 6, that in the case of FastQS, only 48% of all trials ended with a mean square root error that was greater than $E = 1e{-}17$. All other trials ended with a perfect reconstruction of the roots. We check consistency of this observation with the error track displayed in Fig. 5.

In comparison, the classical ClosedQS produced an error greater than $E = 1e{-}17$ in approximately 95% of all cases. CompQS, finally, produced a mean square root error greater than $E = 1e{-}17$ in 100% of all trials. More than that: We can see that approximately 97% of the computed roots had a mean square root error greater than $E = 1e{-}15$. Besides the extreme runtimes of this algorithm, this dramatic loss of accuracy is probably an inherent characteristics of the underlying unsymmetric QR algorithm of the `dgeev.f` subroutine. Similar observations were reported by J.W. Demmel in the case of the implicit tridiagonal QR algorithm [9]. There is some kind of a "barrier" in these algorithms that prevents them from reaching optimal accuracies. This is clearly seen from our experiments because now, we have a reference algorithm in terms of FastQS at our disposition that does an incomparably better job.

Fig. 7 displays the number of iterations required in the tandem backward optimizer of FastQS. We can see that in over 450 000 trials, the backward optimizer converged after a single iteration. The necessary number of iterations decreases drastically. We see that only 10% of all trials required 4 iterations or more than 4 iterations to converge. In all experiments,

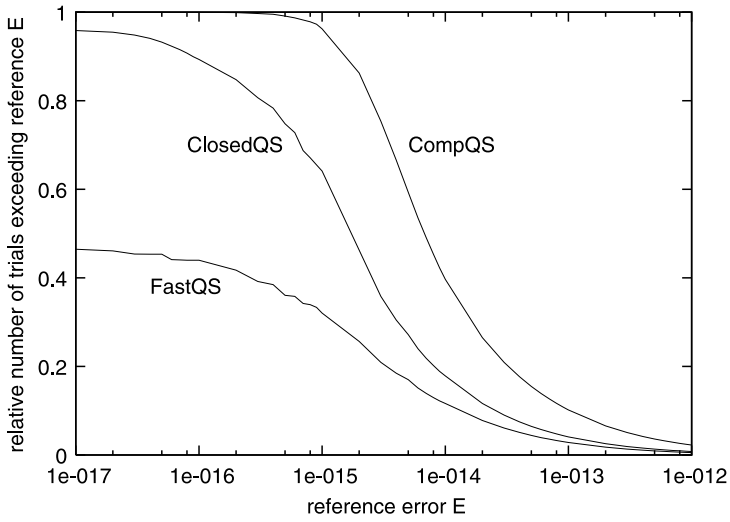

Fig. 6. Probability that a mean square root error exceeds a prescribed reference error level E in the case of 4 real roots. ClosedQS: Closed-form solver. CompQS: Companion/eigenvalue solver. FastQS: Fast quartic solver.

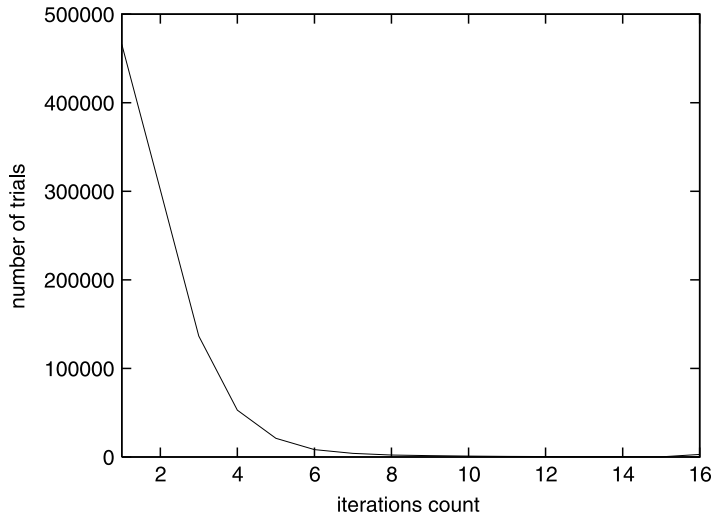

Fig. 7. Number of trial runs that terminate after a certain number of iterations of the quartic-to-quadratic backward optimizer in one million trials in the case of 4 real roots.

the maximum iterations counter was set to a value of $m = 16$. A careful inspection of the plot in Fig. 7 reveals that we have an almost invisible number of trials that ran up to this maximum iterations count. Approximately 0.2% of all trials passed through our limit cycle detector without being classified as converged in terms of this criterion. In fact, the criterion is very strict, and therefore, some limit cycles are not caught by the test because of minor numerical mismatch. Still all of these unclassified cases had converged, as our detailed inspection revealed. Notice that one secret behind the fabulous accuracies reached by the FastQS algorithm is the fact that this algorithm is completely threshold-free.

Table 4 shows the overall runtimes of the 4 algorithms under comparison for one million trials of this all real root experiment. These runtimes were acquired using the intrinsic subroutine `cpu_time` under Fortran. Calls on this subroutine were placed right before and right after the respective subroutine calls and the resulting execution times were accumulated over one million trial runs. We ran 5 experiments of one million trials each with the 4 algorithms under test to determine these runtimes as shown in Table 4. We can see the dramatic difference in runtime between the unsymmetric QR eigenvalue based methods on the one side, and ClosedQS, FastQS on the other side. The additional refinements in FastQS are computationally quite efficient, as we can see. They increase the overall runtime approximately by a factor of 2 over the closed-form solution algorithm. Moreover, we observe that these runtimes vary from experiment to experiment. The reasons for this effect are unknown. The test algorithms are executed under the Windows XP operation system. Moreover, we observed that all these runtimes can grow by some factor when the subroutines are called from larger simulation programs.

Table 4
Runtimes (in s) for the 4 quartic solvers in 5 independent experiments comprising one million trials per experiment.

Experiment no.	1	2	3	4	5
CompQS	26.75	26.59	26.73	27.11	26.41
C02alf.f	35.64	35.20	35.32	35.34	35.42
ClosedQS	2.03	2.04	2.01	2.26	2.20
FastQS	3.90	3.81	3.64	3.90	3.53

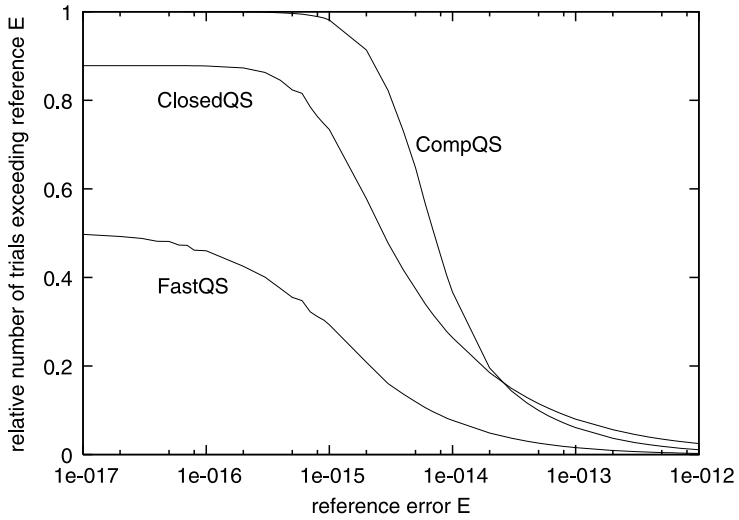

Fig. 8. Probability that a mean square root error exceeds a prescribed reference error level E in the case of 2 real roots and one complex conjugate root pair. ClosedQS: Closed-form solver. CompQS: Companion/eigenvalue solver. FastQS: Fast quartic solver.

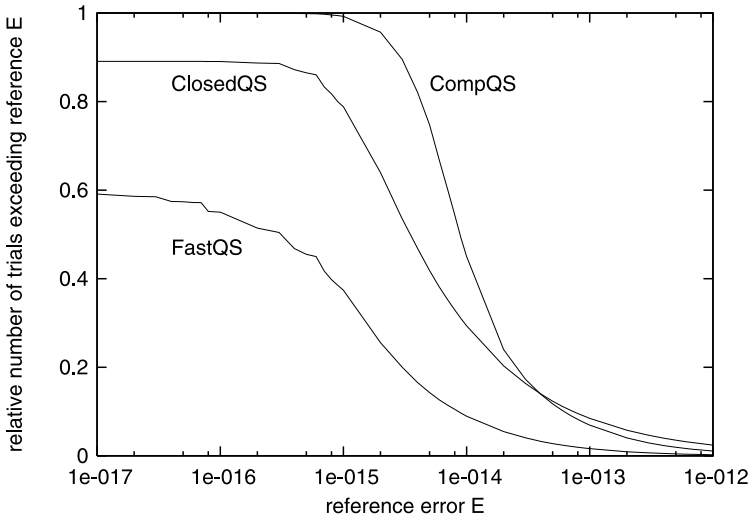

Fig. 9. Probability that a mean square root error exceeds a prescribed reference error level E in the case of two complex conjugate root pairs. ClosedQS: Closed-form solver. CompQS: Companion/eigenvalue solver. FastQS: Fast quartic solver.

These experimental evaluations are repeated with root configurations (52) and (53). The results of the statistical evaluations are displayed in Figs. 8 and 9, respectively. We can see that these statistics appear quite similar to the results obtained in the all real root case of Fig. 6.

The CompQS algorithm performs practically identical in all root cases. This is perhaps one of the reasons why this algorithm is often regarded as highly reliable.

The ClosedQS algorithm produces an increased number of perfect results, where the roots are exactly reconstructed without any error, in about 10% of all cases. On the other hand, if there is some error, then it is immediately greater than $E = 2e{-}16$, approximately. This can be seen from the flat statistics of ClosedQS in the range $1e{-}17 < E < 1e{-}15$ in Figs. 8 and 9.

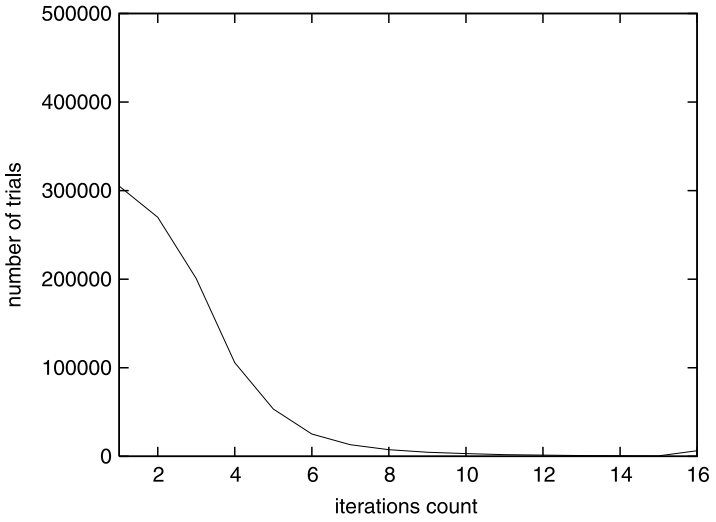

Fig. 10. Number of trial runs that terminate after a certain number of iterations of the quartic-to-quadratic backward optimizer in one million trials in the case of two complex conjugate root pairs.

A superior performance is again confirmed for the FastQS algorithm, which performs significantly better than the other algorithms in the test. We observe a slightly decreased probability of perfectly reconstructed roots in the all complex roots experiment of Fig. 9.

Fig. 10 shows the number of iterations required by the tandem backward optimizer in the case of the all complex roots experiment. We can compare this with the iterations counts as displayed in Fig. 7 for the case of the all real root experiment. Apparently, the number of iterations increases and the all complex root case requires more effort on the side of the refinement algorithm. An even more detailed inspection of the plot in Fig. 10 reveals that in this case, we counted approximately 0.6% of all trials that passed through out limit cycle test without being detected although all these cases had converged.

4.2. Some individual tests

Let us continue with some individual experiments. Those are devoted to the extreme spread characteristics of the algorithms. Sometimes the "robustness" of the algorithms is quoted in terms of their capability to reconstruct root sets with extreme spread. We will now show some instructive results from experiments of this kind, which we conducted with spreads ranging up to $S = 1e18$ according to the definition of the spread as given in (2).

Consider Table 5, where we show the results of an experiment with all real roots, increasing at a dramatic rate: $x_1 = 1$, $x_2 = 1e6$, $x_3 = 1e12$ and $x_4 = 1e18$. The results of the three algorithms under comparison show dramatic differences: ClosedQS reconstructs the dominant root perfectly, reconstructs the first subdominant root with a large error, and fails completely in producing any reasonable result for the two "tiny" roots. Clearly, this is a consequence of the twofold squaring of coefficients that is inherent with the closed-form solution formulas as a consequence of the necessary depression transformation underlying the classical closed-form solution formulas. Notice also that ClosedQS produces the roots in disorder.

The third block in Table 5 shows the results obtained from CompQS. We confirm a convincing performance. Only the estimate of x_2 appears with two deteriorated mantissa digits. The other roots are perfectly reconstructed.

Finally, we see the result obtained with FastQS in block 4 of Table 5. This algorithm comes up with the best result because all roots are perfectly reconstructed. Moreover, we recall that this is the result of the backward optimizer (Table 1) initialized with the two dominant roots of ClosedQS. Apparently, the fact that the dominant root is perfectly reconstructed by the closed-form solver is the key why the backward optimizer converged under these circumstances, in view of the heavily disturbed first subdominant root estimate of the closed-form solver. The two deteriorated tiny roots are not used by the backward optimizer. They are reconstructed by the fast least squares algorithm of Table 3 from the quartic coefficients and the much more reliable dominant roots.

Continue with an inspection of the next experiment as displayed in Table 6. Here we see a case with two large real roots and one small complex conjugate root pair with extremely small imaginary part. This should be even more difficult to identify by the algorithms. In fact, ClosedQS produces the expected inferior characteristics on the tiny roots, but again exhibits the important characteristics that the dominant root is reconstructed perfectly without any error. This is of key importance for our backward optimizer, because such an initial estimate will safely fall into the basin of attraction of the underlying fixed-point iteration. Indeed, the table confirms a fascinating result for the FastQS algorithm. We loose only two digits in the representation of the extremely tiny imaginary parts of the tiny complex conjugate root pair.

Table 5
Extreme spread test no. 1: Linearly ascending real roots.

Exact roots	(1.00000000000000, 0.000000000000000E+000) (1000000.00000000, 0.000000000000000E+000) (1000000000000.00, 0.000000000000000E+000) (1.000000000000000E+018, 0.000000000000000E+000)
ClosedQS	(1.000000000000000E+018, 0.000000000000000E+000) (−344251184960.000, 1021787072282.58) (1688503369984.00, 0.000000000000000E+000) (−344251184960.000, −1021787072282.58)
CompQS	(1.00000000000000, 0.000000000000000E+000) (999999.999999985, 0.000000000000000E+000) (1000000000000.00, 0.000000000000000E+000) (1.000000000000000E+018, 0.000000000000000E+000)
FastQS	(1.000000000000000E+018, 0.000000000000000E+000) (1000000000000.00, 0.000000000000000E+000) (1000000.00000000, 0.000000000000000E+000) (1.00000000000000, 0.000000000000000E+000)

Table 6
Extreme spread test no. 2: Two large real roots and one small complex conjugate root pair with very small imaginary parts.

Exact roots	(10.0000000000000, 0.100000000000000) (10.0000000000000, −0.100000000000000) (1000000.00000000, 0.000000000000000E+000) (1000000000000.00, 0.000000000000000E+000)
ClosedQS	(1000000000000.00, 5.820766091346741E−011) (957885.235351562, 914784.381435767) (957886.621551514, −914783.704644411) (−915751.856872559, −0.676791356352624)
CompQS	(10.0000000000026, 9.999999972441263E−002) (10.0000000000026, −9.999999972441263E−002) (1000000.00000000, 0.000000000000000E+000) (1000000000000.00, 0.000000000000000E+000)
FastQS	(1000000000000.00, 0.000000000000000E+000) (1000000.00000000, 0.000000000000000E+000) (10.0000000000000, 0.100000000000026) (10.0000000000000, −0.100000000000026)

We check this with the result obtained with CompQS. Here, we can see that these tiny imaginary parts of the tiny complex conjugate root pair appears heavily corrupted by roundoff error. This unsymmetric QR concept underlying the CompQS algorithm looses 8 significant mantissa digits.

We shall continue with one more experiment of this extreme spread kind. Table 7 shows the results of this test with one very small and one very large real root, and an intermediate complex conjugate root pair with relatively small imaginary part.

Again, we can confirm this typical characteristics that ClosedQS reconstructs the dominant root perfectly. The subdominant roots appear largely deteriorated. Notice that in this case, the backward optimizer in FastQS cannot benefit from this perfectly reconstructed dominant real root, because it is directly followed by a complex conjugate root pair in the magnitude ranking according to Table 2. Therefore, initialization set 1 (37a), (37b) will not represent a valid quadratic with real coefficients. On the other hand, initialization set 2 (38a), (38b) will be filled with the largely destroyed complex conjugate root pair. Hence it should be particularly hard for the backward optimizer in FastQS to handle such a situation.

But the results in Table 7 confirm that the algorithm converged and produced a brilliant overall reconstruction of the roots. We loose only 4 mantissa digits in the relatively small imaginary parts of the complex conjugate root pair. Both real roots are perfectly reconstructed.

We compare this result with the result of CompQS as shown in block 3 of Table 7. We see that CompQS also looses 4 digits on the small imaginary parts, but additionally looses 5 significant digits on the tiny real root, while the same tiny real root was perfectly reconstructed by FastQS.

Apparently, this test was particularly difficult for the FastQS algorithm, because the perfect dominant root results in no valid initialization, because of a ranking conflict. The only valid initialization was the deteriorated complex conjugate root pair, as provided by the closed-form solver.

Table 7
Extreme spread test no. 3: One very small and one very large real root, plus one intermediate complex conjugate root pair with relatively small imaginary parts.

Exact roots	(1.00000000000000, 0.000000000000000E+000) (10000.0000000000, 100.000000000000) (10000.0000000000, −100.000000000000) (10000000000000.0, 0.000000000000000E+000)
ClosedQS	(10000000000000.0, 0.000000000000000E+000) (6667.00000000000, 32768.0000000000) (6667.00000000000, 0.000000000000000E+000) (6667.00000000000, −32768.0000000000)
CompQS	(0.999999999978741, 0.000000000000000E+000) (9999.99999999998, 100.000000004217) (9999.99999999998, −100.000000004217) (10000000000000.0, 0.000000000000000E+000)
FastQS	(10000.0000000000, 99.9999999998510) (10000.0000000000, −99.9999999998510) (10000000000000.0, 0.000000000000000E+000) (1.00000000000000, 0.000000000000000E+000)

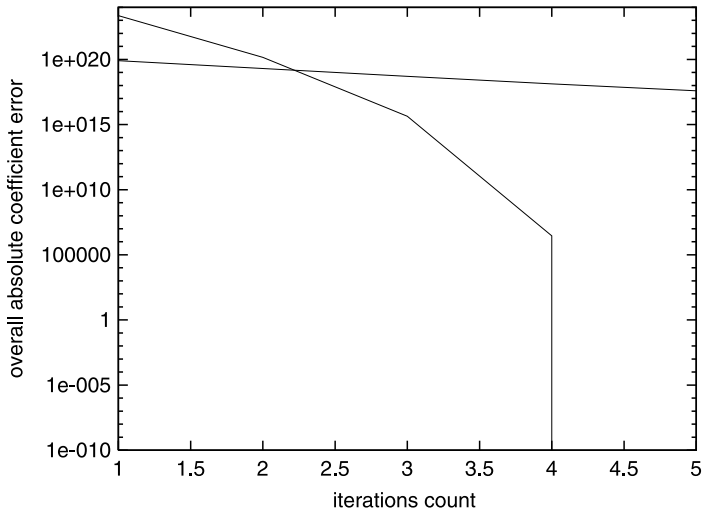

Fig. 11. Convergence characteristics for extreme spread test no 3. Overall absolute coefficient error ϵ displayed for the two tandem iterations. Tandem iteration 2 converges with a perfect null in iteration 5.

The results show that we were still safely inside the basin of attraction with this initial guess. This confirms again the large basin of attraction capability of this underlying fixed-point iteration, which is a consequence of the fact that the quartic coefficient error vector (5) is only a weakly nonlinear function of the $\{\alpha, \beta, \gamma, \delta\}$ parameters.

A detailed inspection of the convergence characteristics of the algorithm in such a difficult case is hence instructive. Fig. 11 shows a plot of the overall absolute coefficient error ϵ (27) as obtained from the two parallel backward refinement iterations in this case. We can see that the first iteration, initialized from the combination (37a) and (37b) converges very slowly. The second iteration, initialized from the relatively deteriorated complex conjugate root estimate, on the other hand, converges rapidly, and drops into a perfect zero ϵ after iteration 5. This appears as a hit of the baseline in this logscale plot at iterations count 4. Notice that even a zero ϵ will not guarantee a perfect reconstruction of the roots. Observe the tremendous dynamic range that is occupied by the ϵ's in this case. This indicates a large distance between the starting points and the true root locations. Finally, it demonstrates the convergence capabilities of the underlying fixed-point iteration.

Let us now come to a final experiment that looks uncritical at first glance. The given roots and the results are displayed in Table 8. We can see here the case of two complex conjugate roots with uncritical spread. Even ClosedQS does a quite acceptable job on the reconstruction of these roots. However, an inspection of the results produced by CompQS reveals that there is an unexpected loss of 6 significant mantissa digits of the small imaginary parts when estimated by this algorithm. FastQS, on the other hand, delivers an almost perfect reconstruction. There appears only a very minor defect in the last digit of the larger imaginary part. This example points at potential difficulties with the unsymmetric QR algorithm in cases where the root spread is particularly small.

Table 8
A less extreme example: Two complex conjugate root pairs with very moderate spread.

Exact roots	(400000.000000000, 300.000000000000) (400000.000000000, −300.000000000000) (30000.0000000000, 7000.00000000000) (30000.0000000000, −7000.00000000000)
ClosedQS	(400000.000000000, −299.999999974569) (30000.0000000000, −6999.99999999891) (400000.000000000, 299.999999974569) (30000.0000000000, 6999.99999999891)
CompQS	(30000.0000000000, 7000.00000000017) (30000.0000000000, −7000.00000000017) (400000.000000000, 300.000000130248) (400000.000000000, −300.000000130248)
FastQS	(400000.000000000, 300.000000000000) (400000.000000000, −300.000000000000) (30000.0000000000, 7000.00000000001) (30000.0000000000, −7000.00000000001)

5. Conclusions

A new algorithm for solving quartic equations has been introduced. This algorithm is a hybrid between a classical closed-form solver and a backward optimizer for iterative root refinement. An unexpected progress in performance and speed has been reached by this algorithm. The background is a weakly nonlinear quadratic-to-quartic coefficient error and some nice properties of the corresponding Jacobian matrix. These properties lead to the design of a particularly streamlined and numerically robust algorithm for a highly accurate fitting of two quadratics onto a given quartic. This optimal solution of the underlying fitting problem is the key to the observed breakthrough both in runtime and in performance.

References

[1] M. Abramowitz, I.A. Stegun, Solutions of quartic equations, Par. 3.8.3, in: Handbook of Mathematical Functions with Formulas, Graps and Mathematical Tables, Dover, New York, 1972, pp. 17–18. 9th printing.
[2] B. Birkhoff, S. Mac Lane, A Survey of Modern Algebra, 5th ed., Macmillan, New York, 1996, pp. 107–108.
[3] Quartic function (by anonymous author), 2009. http://en.wikipedia.org/wiki/Quartic_function.
[4] G.H. Golub, C.F. Van Loan, Matrix Computations, second ed., John Hopkins University Press, Baltimore, MD, 1989.
[5] E. Anderson, et al., LAPACK User's Guide, third ed., SIAM Customer Service, Pittsburgh, PA, 1999.
[6] NAG Fortran Library Manual, Mark 21, NAG LTD, online version, 1999.
[7] R.L. Ward, Quartic equations, The Math Forum@Drexel, 2009. http://mathfo-rum.org/dr.math/faq/faq.cubic.equations.html.
[8] W.H. Press, Numerical Recipes in FORTRAN 77: The Art of Scientific Computing, second ed., Cambridge University Press, Cambridge, MA, 1992.
[9] J.W. Demmel, The inherent inaccuracy of implicit tridiagonal QR, IMA Preprint Series # 963, April 1992.